Automatic Mapping of Parallel Applications on Multicore Architectures using the Servet Benchmark Suite

Jorge González-Domínguez*, Guillermo L. Taboada, Basilio B. Fraguela, María J. Martín, Juan Touriño

Computer Architecture Group, Department of Electronics and Systems, University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain

Abstract

Servet is a suite of benchmarks focused on detecting a set of parameters with high influence on the overall performance of multicore systems. These parameters can be used for autotuning codes to increase their performance on multicore clusters. Although Servet has been proved to detect accurately cache hierarchies, bandwidths and bottlenecks in memory accesses, as well as the communication overhead among cores, up to now the impact of the use of this information on application performance optimization has not been assessed. This paper presents a novel algorithm that automatically uses Servet for mapping parallel applications on multicore systems and analyzes its impact on three testbeds using three different parallel programming models: message-passing, shared memory and PGAS (Partitioned Global Address Space). Our results show that a suitable mapping policy based on the data provided by this tool can significantly improve the performance of parallel applications without source code modification.

Keywords: Mapping Policies, Multicore Architectures, Parallel Benchmarks, Performance Evaluation

*Principal corresponding author: Jorge González-Domínguez; Tel.: +34-981-167000 (Ext 1376); fax: +34-981-167160
Email addresses: jgonzalezd@udc.es (Jorge González-Domínguez), taboada@udc.es (Guillermo L. Taboada), basilio.fraguela@udc.es (Basilio B. Fraguela), mariam@udc.es (María J. Martín), juan@udc.es (Juan Touriño)

Preprint submitted to Computers and Electrical Engineering. Special Issue: Emerging Computing Architectures and Systems November 25, 2011

1. Introduction

The popularity of autotuned codes, which adapt their behavior to the machine where they are executed, has increased during the last years thanks to the performance improvement that they are able to achieve. A widespread autotuning technique for sequential codes consists in using a wide search mechanism to find the most suitable algorithm [1, 2, 3]. The search time could be reduced by knowing the values of some parameters of the system [4, 5].

As regards parallel codes, there exists in the literature many optimization techniques to improve their performance, most of them directed toward increasing the communication bandwidth among the processes [6, 7, 8]. Among the different parallel architectures, clusters of multicores are nowadays the target architecture for autotuned codes. On the one hand, they usually present several non-uniform latencies and bandwidths depending on the cores that are communicating [9]. On the other hand, the overall memory access throughput might decrease if several cores share memory or cache.

In this situation some knowledge of the memory system topology, as well as some hardware parameters, are required for every optimization effort. However, the system parameters and specifications are usually vendor-dependent and often inaccessible to user applications. Therefore, the estimation by benchmarks is the only general and portable way to find out the hardware characteristics without worrying about the vendor, the operating system (OS) or the user privileges. Besides, this approach provides experimental results about the performance of the systems, obtaining a more reliable estimate than inferring them from the machine specifications. Servet [10, 11] is a portable benchmark suite designed to obtain the relevant hardware parameters of clusters of multicores and, hence, support the automatic optimization of parallel codes on these architectures. Cache size and hierarchy, bottlenecks in memory access and communication overheads are included among the estimated parameters.

This paper presents a novel algorithm to automatically generate an efficient mapping policy of parallel applications on multicore systems according to the information provided by Servet and without source code modifications. Furthermore, an analysis of the impact of the use of these mapping policies in the performance optimization of benchmark applications is provided. This study is based on the performance results obtained by a widespread parallel benchmark suite, the NAS Parallel Benchmarks (NPB) [12], using their implementations for Message Passing Interface (MPI) [13], Open Multi-Processing (OpenMP) [14] and Unified Parallel C (UPC) [15]. The latter UPC is a Partitioned Global Address Space (PGAS) language which is an emerging alternative for programming hybrid shared/distributed memory multicore architectures. The tests are run in three systems with different architectures: two clusters of multicores (hybrid shared/distributed memory systems) and a shared-memory multiprocessor (SMP) machine. Besides, an analysis of the increase of performance in the two clusters when executing benchmarks with a hybrid MPI+OpenMP implementation (the NPB-MZ kernels) is also included.

The rest of the paper is organized as follows. Section 2 presents previous works focused on measuring the impact of architecture-aware optimization techniques, with a special focus on the evaluation of mapping proposals. Section 3 summarizes the benchmarks developed in Servet and explains the algorithm to provide the automated mapping policy. Section 4 describes the experimental testbeds, presents the performance results and analyzes the impact of the use of Servet on the performance achieved. Finally, concluding remarks are presented in Section 5.

2. Related Work

Two main approaches are followed to improve the performance of parallel applications in clusters of multicores. The most common one consists in implementing and timing several codes in order to choose the best one according to the system characteristics, for instance, adapting the communication algorithms to the target machine. In [16] Vadhiyar et al. present a thorough evaluation of the MPI collectives that proves that the optimal algorithm and the optimal buffer size for a given message size depends on the gap values of the networks, the memory models and the underlying communication layer. The optimal parameters for a particular system are experimentally determined. Faraj et al. [17, 18] present an automatic generation and tuning system for MPI collective communication routines and offer a successful evaluation of its impact on the NAS Parallel Benchmarks. Their approach focuses on optimizing collectives taking into account the network topology.

The second approach consists in assigning processes or threads to specific cores to improve the performance without source code modifications. In [19] Chen et al. propose a profile-guided approach for optimizing parallel process placement in SMP clusters, experimentally proving that it can obtain a good speed-up. However their approach only considers the communication costs, and the experimental tests are limited to MPI. Mercier and Clêt-Ortega [20] show another evaluation restricted to MPI for several mapping policies using the NPB benchmarks. This work also includes a study about the influence of the shared caches on the mapping policies. However, the placement technique is only based on the global amount of data exchanged and it does not rely on real hardware parameters, but on assumptions about the communication costs. Besides, the tests are only run for the NPB with a high amount of irregular communications and they only cover the message-passing model. In [21] Broquedis et al. propose a hierarchical approach to the execution of OpenMP threads on multicore machines, providing multicore-aware and memory-aware scheduling policies. Nevertheless, in order to create the mapping policy, this approach deduces the communication and memory overheads from the knowledge of the total number of nodes, the number of cores in each node and the cache topology instead of experimentally measuring them. Besides, it relies on the tool hwloc [22], which can only obtain the topology of the machine when available from the system specifications.

The Servet benchmark suite [10] reports detailed information on hardware parameters of clusters of

multicores which are relevant for performance optimization, such as cache size and hierarchy, bottlenecks in memory access and communication overheads. Thus, the information that it provides supports the selection of the mapping policy, reducing the network overhead and maximizing the concurrent memory access throughput. Nevertheless, the analysis of the impact of the use of Servet on applications performance has not been covered in [10].

This paper improves previous works by: (1) developing an algorithm to automatically generate the mapping; (2) basing these mapping policies not only on the communication costs but also on minimizing the memory access overhead; (3) providing an analysis of the impact of mapping policies based on the hardware estimations of Servet, a portable and publicly available tool; (4) using three testbeds with different architectures (two clusters of multicores and one SMP machine) for assessing the impact of the use of the information provided by Servet; (5) including studies about the impact of the mapping policies on MPI, OpenMP and UPC, as representative programming models for multicore systems: message-passing, shared memory and PGAS, respectively; and (6) adding an analysis of the impact of Servet on hybrid MPI+OpenMP applications.

3. Servet Benchmark Suite for Automatic Mapping on Multicore Architectures

Servet is a fully portable suite of benchmarks to obtain the most important hardware parameters to support the automatic optimization of parallel applications on multicore clusters. These benchmarks determine the number of cache levels, their sizes, the cache topology of shared caches, the memory access bottlenecks, and the communications scalability and overheads. The tests in [10] prove that the suite provides highly accurate estimates according to the system architecture specifications.

Many optimization techniques can take advantage of the hardware parameters determined by Servet. For instance, the portable estimation of the cache sizes allows the use of tiling, one of the best known optimization techniques. Furthermore, many programs provide several implementations of parts of their code in order to take advantage of different architectures. Using the system parameters obtained by Servet it is possible to automatically select the best option to maximize the performance of the application. These results can also be useful for modeling the performance of heterogeneous clusters [23].

Moreover, the information about the overheads can be used to automatically map the processes or threads to certain cores in order to avoid either communication or memory access bottlenecks. Even if not all the overheads can be avoided, mapping policies that minimize their impact can be applied. The potential performance benefits of the use of the information provided by Servet for mapping issues have motivated the development of a novel algorithm that automatically provides the placement of processes or threads to specific cores. The mapping is chosen based on the information about shared caches, overheads in the access to memory and communication layers provided by Servet according to Algorithm 1. Each core in the system

is assigned a weight that represents the overhead cost of its selection. Initially, all the weights are 0, which means that any core can be selected. From then on, the core with the lowest weight is chosen. Whenever a core is selected, the weights are updated according to the following rules:

1. The weights of the cores that share cache with the selected one are increased. This rule is applied for each cache level to avoid the loss of performance when shared caches lead to an increase of the number of cache misses.
2. The weights of the cores that show additional overhead when accessing memory concurrently with the selected core are increased.
3. The weights of the cores whose communication latencies with the selected one are significantly lower than the maximum latency in the system (*MaxLatency*, obtained by Servet) are decreased to promote their selection. They are decreased in a magnitude that depends on the difference between the current latency and *MaxLatency*. Note that shared memory transfer optimizations in the communication libraries can be taken into account through the application of this rule.

The increase or decrease applied to each one of the previous rules depends on the characterization of the code as either memory bound or communication intensive. Currently this characterization is provided by the user through the SERVET_MEM_PRIOR or SERVET_COMM_PRIOR parameters, respectively. A mechanism to automatically categorize each application is expected to be developed in a near future. In a memory bound code the increase due to rules 1 and 2 is ten times larger than the decrease applied by rule 3. In a communication intensive code the opposite practice is applied.

The operation of this mapping procedure is illustrated through a 2-node x86_64 multicore cluster with InfiniBand (20 Gbps) as interconnection network. Each node has 2 Intel Xeon Nehalem quadcore E5520 CPUs at 2.27 GHz and 8 GBytes of memory. Servet has detected on this system the correct cache sizes (L1 32 KBytes; L2 256 KBytes; L3 8 MBytes) and topology, where the L3 cache is the only one shared, by pairs of cores. Moreover, the simultaneous access to memory by two cores within the same processor presents an important overhead. Finally, the latency of inter-node communications is significantly higher than the intra-node ones. Figure 1 presents the architecture of this system. Tables 1 and 2 present step by step the operation of the selection procedure (following Algorithm 1) for mapping 4 processes to this system with a SERVET_MEM_PRIOR and a SERVET_COMM_PRIOR policy, respectively.

The core selected in each iteration is in boldface in the tables. In Table 1, initially core 0 is selected. After mapping the first process, the second iteration starts applying the first rule, which increases the value of the core 2, as it shares L3 cache with core 0. L1 and L2 caches are not considered because they are not shared. Then, rule 2 increases the weights of cores 2, 4 and 6, because they present an overhead when accessing memory concurrently with core 0. Finally, using the third rule, the weights of the cores in the same node, whose communications are faster, are decreased. The increase due to rules 1 and 2 is ten times

```
foreach core c in the system do
  | Weight[c] = 0
end
foreach process p to map do
  | Assign p to the core c with the lowest Weight[c]
  | // Update the weights of the cores
  | foreach cache in the system do
  |   | foreach core c2 that shares that cache with c do
  |   |   | Increase Weight[c2]
  |   | end
  | end
  | foreach memory access overhead in the system do
  |   | foreach core c2 that shares the overhead with c do
  |   |   | Increase Weight[c2]
  |   | end
  | end
  | foreach core c2 still not assigned do
  |   | if latency(c,c2) < MaxLatency then
  |   |   | Decrease Weight[c2]
  |   | end
  | end
end
```

Algorithm 1: Algorithm to provide the best mapping policy

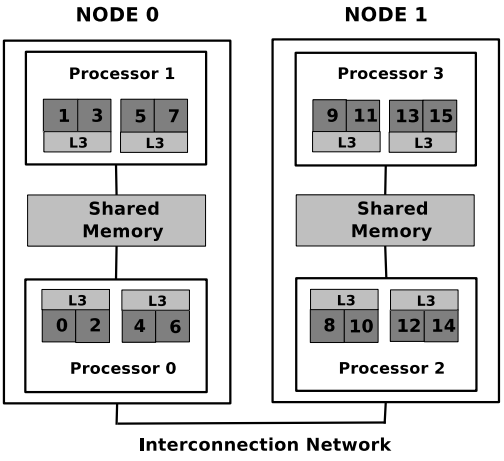

Figure 1: Architecture of 2 nodes of the x86_64 cluster

higher than the decrease considered for the third rule because SERVET_MEM_PRIOR was selected. Once

all the rules have been applied, core 1 is selected because it is the first core with the lowest weight value. With this selection the messages between both cores present low latency (according to Servet, there is no significant difference in the intra-node latencies) and all the memory access overheads are avoided. As can be seen in Table 1, the algorithm leads to map the processes to cores 0, 1, 8 and 9. Although there would be some messages in the interconnection network because both nodes are used, this assignment reduces memory access overhead.

↓ Iter.	Core →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Total	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Rule 1	-	0	*10*	0	0	0	0	0	0	0	0	0	0	0	0	0
	Rule 2	-	0	*10*	0	*10*	0	*10*	0	0	0	0	0	0	0	0	0
	Rule 3	-	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	0	0	0	0	0	0	0	0
	Total	-	***-1***	*19*	*-1*	*9*	*-1*	*9*	*-1*	0	0	0	0	0	0	0	0
3	Rule 1	-	-	0	*10*	0	0	0	0	0	0	0	0	0	0	0	0
	Rule 2	-	-	0	*10*	0	*10*	0	*10*	0	0	0	0	0	0	0	0
	Rule 3	-	-	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	0	0	0	0	0	0	0	0
	Total	-	-	*18*	*18*	*8*	*8*	*8*	*8*	**0**	0	0	0	0	0	0	0
4	Rule 1	-	-	0	0	0	0	0	0	-	0	10	0	0	0	0	0
	Rule 2	-	-	0	0	0	0	0	0	-	0	*10*	0	*10*	0	*10*	0
	Rule 3	-	-	0	0	0	0	0	0	-	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*
	Total	-	-	*18*	*18*	*8*	*8*	*8*	*8*	-	***-1***	*19*	*-1*	*9*	*-1*	*9*	*-1*

Table 1: Evolution of the weight values for all cores in two nodes of the x86_64 cluster when mapping 4 processes with SERVET_MEM_PRIOR

Table 2 shows the evolution of the weights for the same example but with SERVET_COMM_PRIOR. Here the algorithm advises to place the processes in cores inside the same node to avoid communications along the network (cores 0, 1, 4 and 5). Besides, inside the node, Servet chooses cores that do not share the L3 cache. However, it cannot avoid the memory overheads caused by concurrent memory accesses from cores in the same processor.

We have experimentally assessed that adjacent threads/processes in shared memory, message-passing and PGAS applications are more likely to communicate among them than the non-adjacent ones. Thus, our proposal assumes this communication pattern when mapping processes to cores, which is a good compromise between an automatic generation of the mapping policy and a manually-driven scheduling process.

↓ Iter.	Core →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Total	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Rule 1	-	0	*1*	0	0	0	0	0	0	0	0	0	0	0	0	0
	Rule 2	-	0	*1*	0	*1*	0	*1*	0	0	0	0	0	0	0	0	0
	Rule 3	-	*-10*	*-10*	*-10*	*-10*	*-10*	*-10*	*-10*	0	0	0	0	0	0	0	0
	Total	-	***-10***	*-8*	*-10*	*-9*	*-10*	*-9*	*-10*	0	0	0	0	0	0	0	0
3	Rule 1	-	-	0	*1*	0	0	0	0	0	0	0	0	0	0	0	0
	Rule 2	-	-	0	*1*	0	*1*	0	*1*	0	0	0	0	0	0	0	0
	Rule 3	-	-	*-10*	*-10*	*-10*	*-10*	*-10*	*-10*	0	0	0	0	0	0	0	0
	Total	-	-	*-18*	*-18*	***-19***	*-19*	*-19*	*-19*	0	0	0	0	0	0	0	0
4	Rule 1	-	-	0	0	-	0	*1*	0	0	0	0	0	0	0	0	0
	Rule 2	-	-	*1*	0	-	0	*1*	0	0	0	0	0	0	0	0	0
	Rule 3	-	-	*-10*	*-10*	-	*-10*	*-10*	*-10*	0	0	0	0	0	0	0	0
	Total	-	-	*-27*	*-28*	-	***-29***	*-27*	*-29*	0	0	0	0	0	0	0	0

Table 2: Evolution of the weight values for all cores in two nodes of the x86_64 cluster when mapping 4 processes with SERVET_COMM_PRIOR

4. Servet-based Performance Optimization

The use of the parameters detected by Servet applied to the automatic mapping can improve significantly the performance of parallel applications. The analysis of the impact of its use on three systems, an x86_64 cluster, an IA64 cluster (the Finis Terrae supercomputer) and an SMP machine, using the MPI, OpenMP and UPC versions of the NAS Parallel Benchmarks, is next presented. Although the experiments do not use all cores/nodes available in the systems (especially in the IA64 cluster), the performance evaluation was carried out on heavy-loaded systems, with 100% of the cores running jobs sent by other users, showing the benefits of the automatic mapping in real scenarios.

A comparison between the performance obtained with the automatic mapping policies indicated by Servet and the default mapping policy for each scenario is provided. The performance metric selected is MOPS (Million of Operations Per Second), although the improvement of performance observed when using Servet is independent of the particular metric (e.g., MOPS, execution times, or GFLOPS) being reported. Prior works generally do not map automatically, requiring performance profiling and source code modification, or even an exhaustive characterization of the behavior of the code. Thus, the only available fair comparison for message-passing, shared memory and PGAS models is the benchmarking of the Servet approach against a default scenario where the mapping policy is defined by the administrators through the manual configuration of the queueing system, based on their experience and wide knowledge both of the system and the applications running on it.

4.1. NAS Parallel Benchmarks description

The NPB consist of a set of kernels and pseudo-applications, taken primarily from Computational Fluid Dynamics (CFD) applications. These benchmarks reflect different kinds of computation and communication patterns that are important across a wide range of applications. This makes them the de facto standard in parallel performance benchmarking. There are NPB implementations available for the main parallel programming languages and libraries. For this study the MPI (NPB-MPI from now on) and OpenMP (NPB-OMP) implementations were chosen as representative of well established programming models, and UPC (NPB-UPC) as representative of an emerging alternative.

There are five benchmarks implemented for all the three languages/libraries studied. The Conjugate Gradient (CG) kernel is an iterative solver that tests regular communications in sparse matrix-vector multiplications. The Fourier Transform (FT) kernel performs series of 1-D Fast Fourier Transforms (FFTs) on a 3-D mesh and it tests aggregated communication performance. Integer Sort (IS) is a large integer sort that evaluates both integer computation performance and the aggregated communication throughput. MultiGrid (MG) is a simplified multigrid kernel that performs both short and long distance communications. The Embarrassingly Parallel (EP) kernel is an embarrassingly parallel code that assesses the floating point performance. Among them, the NPB evaluated in this work are: CG, FT, IS and MG. EP was not tested because the default version (without using Servet) obtains efficiencies close to 100%, as shown in [24], and therefore there is no room for improvement. Each kernel has several workloads to scale from small systems to supercomputers. In order to test the benchmarks with the highest possible workload, the C problem size was initially selected. However, because of memory constraints in NPB-UPC, the B size was used for FT and MG.

The memory footprint is the main performance bottleneck in the tests, so the SERVET_MEM_PRIOR policy has been chosen. The exception is UPC MG, which presents continuous communications with different characteristics (among different number of processes and with different message sizes). Therefore, the performance of this kernel depends on the quality of the underlying communications available for the test. As UPC communications are not as optimized as those of MPI and OpenMP the SERVET_COMM_PRIOR policy has been selected in the UPC MG case.

4.2. Servet and NPB on an x86_64 cluster

The first testbed used for the performance evaluation of the impact of Servet on the NPB is a small departmental x86_64 cluster with 8 nodes as those presented in Figure 1. As mentioned before, each node has 2 Intel Xeon Nehalem quadcore E5520 CPUs at 2.27 GHz and 8 GBytes of memory. Besides, the OS is CentOS Linux 5.3, the compilers are the Intel C and Fortran (icc and ifort) v11.1.059 with the optimization flag -O3, the MPI library is Intel MPI (impi) v4.0.0.017 and the UPC compiler/runtime is Berkeley UPC v2.10.0.

The left graphs of Figure 2 present the performance obtained by the NPB in the x86_64 cluster, both with the default scenario and using Servet. The right graphs show the percentage increase of MOPS that Servet achieves with respect to the baseline. It is measured as the difference of the MOPS obtained with Servet and without it, divided by the MOPS obtained in the default scenario and multiplied by 100. In the default scenario the queueing system schedules the tasks using the minimum number of nodes and the operating system automatically maps the threads to specific cores within each node. The use of Servet involves the use of the automatic mapping that it provides. If the SERVET_MEM_PRIOR policy is specified the memory access overhead is minimized through: 1) using only one core per processor for experiments up to 16 cores; and 2) using only one core per L3 cache for experiments on 32 cores. If the SERVET_COMM_PRIOR policy is specified (as mentioned before, it is only used for UPC MG) the algorithm focuses on minimizing the inter-node messages by assigning the threads to cores inside the same node, and, within the node, the threads are mapped trying to minimize the memory access overhead.

The results, only for MPI and UPC, as this cluster is a distributed memory architecture, show that Servet allows the NPB to improve their performance results in all the cases, especially for MPI. The highest MPI performance benefits have been obtained for FT, achieving over 85% of performance improvement for 16 cores. For UPC the best result is for the IS kernel with 32 cores (60% of performance improvement). A study about the improvement of the performance of the NPB-OMP on a large SMP machine will be shown in Section 4.4.

4.3. Servet and NPB on an IA64 cluster

The impact of Servet on NPB performance has also been tested on the Finis Terrae supercomputer. It consists of 142 nodes, each of them with 8 Montvale Itanium2 (IA64) dual-core processors at 1.6 GHz (16 cores per node), 128 GBytes of memory and InfiniBand (20 Gbps) as interconnection network. Each node is divided in two cells with 4 dual-core processors each, where two processors (hence 4 cores) share the memory access bus. The Finis Terrae software configuration consists of a SuSE Linux Enterprise Server 10 IA64 OS, the Intel C and Fortran compilers (icc and ifort) v10.1.012 with OpenMP support and all the optimizations enabled (-O3 flag), HP-MPI v2.2.5.2 and Berkeley UPC v2.8.0.

As presented in [10], Servet, besides obtaining the actual cache sizes (L1 16 KBytes; L2 256 KBytes; L3 9 MBytes) and hierarchy (all caches are private), also characterizes correctly the memory access overhead. In this system, if two cores share the memory access bus, they can only achieve around 1000 MBytes/s memory bandwidth when accessing simultaneously, whereas if two cores are in the same cell but do not share the bus, the bandwidth obtained is approximately 1700 MBytes/s. Finally, if the two cores are located in different cells, they achieve up to 2200 MBytes/s. As for the communications, intra-node transfers are around twice faster than inter-node ones.

Figure 3 presents the impact on the NPB performance of the mapping provided by Servet using up

to 128 cores. Here the default mapping policy is to maximize the number of cores per node in order to minimize inter-node communication, and then, let the OS control the assignment of threads/processes to specific cores. In the Servet mapping if the SERVET_MEM_PRIOR policy is selected the memory access performance is maximized using only one core per memory access.

The results, only obtained for distributed memory programming models, show that Servet clearly improves the performance of NPB codes, especially for MPI. Moreover, the performance advantages of Servet usually increase with the number of cores, improving significantly MPI and UPC scalability. In fact, the use of Servet even allows MPI and UPC running on 64 cores to outperform some NPB original results (without using Servet) on 128 cores. As for UPC MG, with the SERVET_COMM_PRIOR policy, there are no important variations of performance because the default scenario also minimizes the number of inter-node communications.

4.4. Servet and NPB on an SMP Superdome

The third testbed is an SMP machine, an HP Superdome with 64 Montvale Itanium2 processors (128 cores) at 1.6 GHz and 1 TByte of shared memory. The 64 processors are distributed in 16 cells with 4 processors each (hence 8 cores per cell), and the memory bus is shared by groups of 4 cores. This system is another node of the Finis Terrae, so the software configuration is the same as presented in Section 4.3. Here Servet has detected the same memory access overheads as in the IA64 nodes.

The main motivation for the use of this system is the analysis of the impact on scalability of the use of Servet on shared memory systems, where only NPB-OMP and NPB-UPC have been tested. The MPI version has not been executed in this machine because the administration policy does not allow it. The use of a single system eases the adoption of a specific mapping policy. As there is only one communication layer, Servet indicates the same placement for both priority options, trying to minimize the concurrent memory access overhead through: (1) using only one core per cell in experiments up to 16 threads; (2) working with only one core per memory bus when using 32 cores, as there are 32 memory buses in this system (2 per cell); and (3) using one core per processor for 64-core experiments. In this latter case, the conflicts on the access to the memory bus cannot be avoided but the mapping policy minimizes them.

Figure 4 shows the NPB results on the Superdome using the OS default mapping policy and the mapping suggested by the parameters obtained by Servet. Once again, the use of Servet presents significant performance benefits, although with less relative importance than for the distributed memory systems (the x86_64 and IA64 clusters). The mapping policy provides noticeable performance advantages, as the use of cores from different cells (application of the Servet policy) outperforms the default OS thread allocation, which tends to map the threads to contiguous cores.

4.5. Servet with several levels of parallelism

The previous sections have demonstrated that Servet can increase the performance of the basic NPB developed for several languages and programming models. However, many important scientific problems feature several levels of parallelism, and this property is not reflected in the initial NPB versions. To address this issue the NPB Multi-Zone (NPB-MZ) take advantage of two-level parallelism through using hybrid MPI+OpenMP codes.

The NPB-MZ include the Lower-Upper Symmetric Gauss-Seidel (LU), Scalar Penta-diagonal (SP), and Block Tri-diagonal (BT) applications, which solve discretized versions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions. Each code operates on a structured discretization mesh that is a logical cube. The flow equations are solved independently in each zone and, after each iteration, the zones exchange boundary values with their immediate neighbors with which they overlap.

Figure 5 shows the performance results for the hybrid NPB-MZ executed with C size in the x86_64 and the IA64 clusters previously described. In all the experiments only one MPI process per node is created, filling the cores inside the nodes with OpenMP threads. The default mapping policy in both systems is the same used in the previous NPB evaluation, minimizing the number of nodes and relying on the OS to place the threads in the specific cores of the nodes. As for Servet, SERVET_MEM_PRIOR is used for the three codes, mapping each thread to one of the cores selected by Servet, as was detailed in Sections 4.2 and 4.3 for the x86_64 and IA64 clusters, respectively. The mapping obtained improves the performance in all the benchmarks, achieving even around 350% performance improvement for the SP application on 128 cores. These results prove that Servet can also be useful for applications that exploit several levels of parallelism through hybrid programming models.

5. Conclusions

This paper has presented an analysis of the impact of the information provided by Servet on the mapping of parallel applications. Servet is a benchmark suite that detects parameters relevant for the performance optimization of applications on multicore architectures, such as the cache sizes and topology, overheads in memory access and communication costs.

An automatic approach to generate a mapping policy to assign parallel processes or threads to cores has been proposed. It uses the parameters detected by Servet and a characterization of the code as either memory bound or communication intensive, provided by the user. The proposed mapping is based on the minimization of the communication cost and the maximization of the memory access throughput, which usually turn to be the limiting factors for the scalability of parallel applications.

An important strength of this work is that it provides a completely automatic mapping, not only because Servet automatically detects the characteristics of the system but also because the mapping policy is gener-

ated for all kinds of applications, without requiring analysis of the source code or application profiling. The main advantages of this general mapping policy are: 1) the user needs very little work to apply it; 2) it can be used in all kinds of applications, implemented with different languages or even programming paradigms and executed on different architectures; 3) the overhead due to the time necessary to analyze the code or profile the application is avoided, also allowing the use of this mapping policy in scenarios where the source code is not available.

Performance results using the widely extended NPB have been presented for three multicore architectures and three different parallel programming models: message-passing (MPI), shared memory (OpenMP) and PGAS (UPC). The NPB-MZ kernels implemented with MPI+OpenMP were also tested. Almost all the results have shown performance advantages with the use of Servet and, most importantly, the use of the proposed mapping strategy does not penalize, in any case, application performance.

The automatic mapping is included in Servet since version 2.0, publicly available under GPL license at `http://servet.des.udc.es`.

Acknowledgments

This work was funded by the Ministry of Science and Innovation of Spain under Project TIN2010-16735 and an FPU grant AP2008-01578, and by the Galician Government (Xunta de Galicia) under Project INCITE08PXIB105161PR. We gratefully thank CESGA (Galicia Supercomputing Center, Santiago de Compostela, Spain) for providing access to the Finis Terrae supercomputer.

References

[1] M. Püschel, J. M. F. Moura, J. Johnson, D. A. Padua, M. M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, N. Rizzolo, SPIRAL: Code Generation for DSP Transforms, Proc. of the IEEE 93 (2) (2005) 232–275.

[2] M. Frigo, S. G. Johnson, The Design and Implementation of FFTW3, Proc. of the IEEE 93 (2) (2005) 216–231.

[3] R. C. Whaley, A. Petitet, J. J. Dongarra, Automated Empirical Optimizations of Software and the ATLAS Project, Parallel Computing 27 (1–2) (2001) 3–35.

[4] K. Yotov, X. Li, G. Ren, M. J. Garzarán, D. A. Padua, K. Pingali, P. Stodghill, Is Search Really Necessary to Generate High Performance BLAS?, Proc. of the IEEE 93 (2) (2005) 358–386.

[5] B. B. Fraguela, Y. Voronenko, M. Püschel, Automatic Tuning of Discrete Fourier Transforms Driven by Analytical Modeling, in: Proc. 18th Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT'09), Raleigh, NC, USA, 2009, pp. 271–280.

[6] J. Zhang, J. Zhai, W. Chen, W. Zheng, Process Mapping for Collective Communications, in: Proc. 15th Euro-Par Conf. (Euro-Par'09), Vol. 5704 of Lecture Notes in Computer Science, Delft, The Netherlands, 2009, pp. 81–92.

[7] S. Sistare, R. Vandevaart, E. Loh, Optimization of MPI Collectives on Clusters of Large-Scale SMPs, in: Proc. 12th ACM/IEEE Conf. on Supercomputing (SC'99), Portland, OR, USA, 1999, pp. 23–36.

[8] V. Tipparaju, J. Nieplocha, D. K. Panda, Fast Collective Operations using Shared and Remote Memory Access Protocols on Clusters, in: Proc. 17th Intl. Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, 2003, pp. 84–93.

[9] E. Musoll, Variable-Size Mosaics: A Process-Variation Aware Technique to Increase the Performance of Tile-Based, Massive Multi-Core Processors, Computers and Electrical Engineering, In Press (2011) DOI: 10.1016/j.compeleceng.2011.05.012.

[10] J. González-Domínguez, G. L. Taboada, B. B. Fraguela, M. J. Martín, J. Touriño, Servet: A Benchmark Suite for Autotuning on Multicore Clusters, in: Proc. 24th Intl. Parallel and Distributed Processing Symposium (IPDPS'10), Atlanta, GA, USA, 2010.

[11] The Servet Benchmark Suite, `http://servet.des.udc.es/` (Last visited: July 2011).

[12] NASA Advanced Computing Division. NAS Parallel Benchmarks, `http://www.nas.nasa.gov/Software/NPB/` (Last visited: July 2011).

[13] Message Passing Interface Forum, `http://www.mpi-forum.org` (Last visited: July 2011).

[14] OpenMP, `http://openmp.org` (Last visited: July 2011).

[15] UPC Consortium: UPC Language, `http://upc.gwu.edu` (Last visited: July 2011).

[16] S. S. Vadhiyar, G. E. Fagg, J. J. Dongarra, Automatically Tuned Collective Communications, in: Proc. 13th ACM/IEEE Conf. on Supercomputing (SC'00), Dallas, TX, USA, 2000, p. 3.

[17] A. Faraj, X. Yuan, Automatic Generation and Tuning of MPI Collective Communication Routines, in: Proc. 19th Intl. Conf. on Supercomputing (ICS'05), Cambridge, MA, USA, 2005, pp. 393–402.

[18] A. Faraj, S. Kumar, B. Smih, A. R. Mamidala, J. A. Gunnels, P. Heidelberger, MPI Collective Communications on the Blue Gene/P Supercomputer: Algorithms and Optimizations, in: Proc. 23rd Intl. Conf. on Supercomputing (ICS'09), Yorktown Heights, NY, USA, 2009, pp. 489–490.

[19] H. Chen, W. Chen, J. Huang, B. Robert, H. Kuhn, MPIPP: An Automatic Profile-guided Parallel Process Placement Toolset for SMP Clusters and Multiclusters, in: Proc. 20th Intl. Conf. on Supercomputing (ICS'06), Cairns, Australia, 2006, pp. 353–360.

[20] G. Mercier, J. Clêt-Ortega, Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments, in: Proc. 16th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'09), Vol. 5759 of Lecture Notes in Computer Science, Espoo, Finland, 2009, pp. 104–115.

[21] F. Broquedis, O. Aumage, B. Goglin, S. Thibault, P.-A. Wacrenier, R. Namyst, Structuring the Execution of OpenMP Applications for Multicore Architectures, in: Proc. 24th Intl. Parallel and Distributed Processing Symposium (IPDPS'10), Atlanta, GA, USA, 2010.

[22] F. Broquedis, J. Clêt-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, R. Namyst, hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications, in: Proc. 18th Euromicro Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP'10), Pisa, Italy, 2010.

[23] B. Javadi, J. H. Abawajy, M. K. Akbari, Performance Modeling and Analysis of Heterogeneous Meta-Computing Systems Interconnection Networks, Computers and Electrical Engineering 34 (6) (2008) 488–502.

[24] D. A. Mallón, G. L. Taboada, C. Teijeiro, J. Touriño, B. B. Fraguela, A. Gómez, R. Doallo, J. C. Mouriño, Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures, in: Proc. 16th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'09), Vol. 5759 of Lecture Notes in Computer Science, Espoo, Finland, 2009, pp. 174–184.

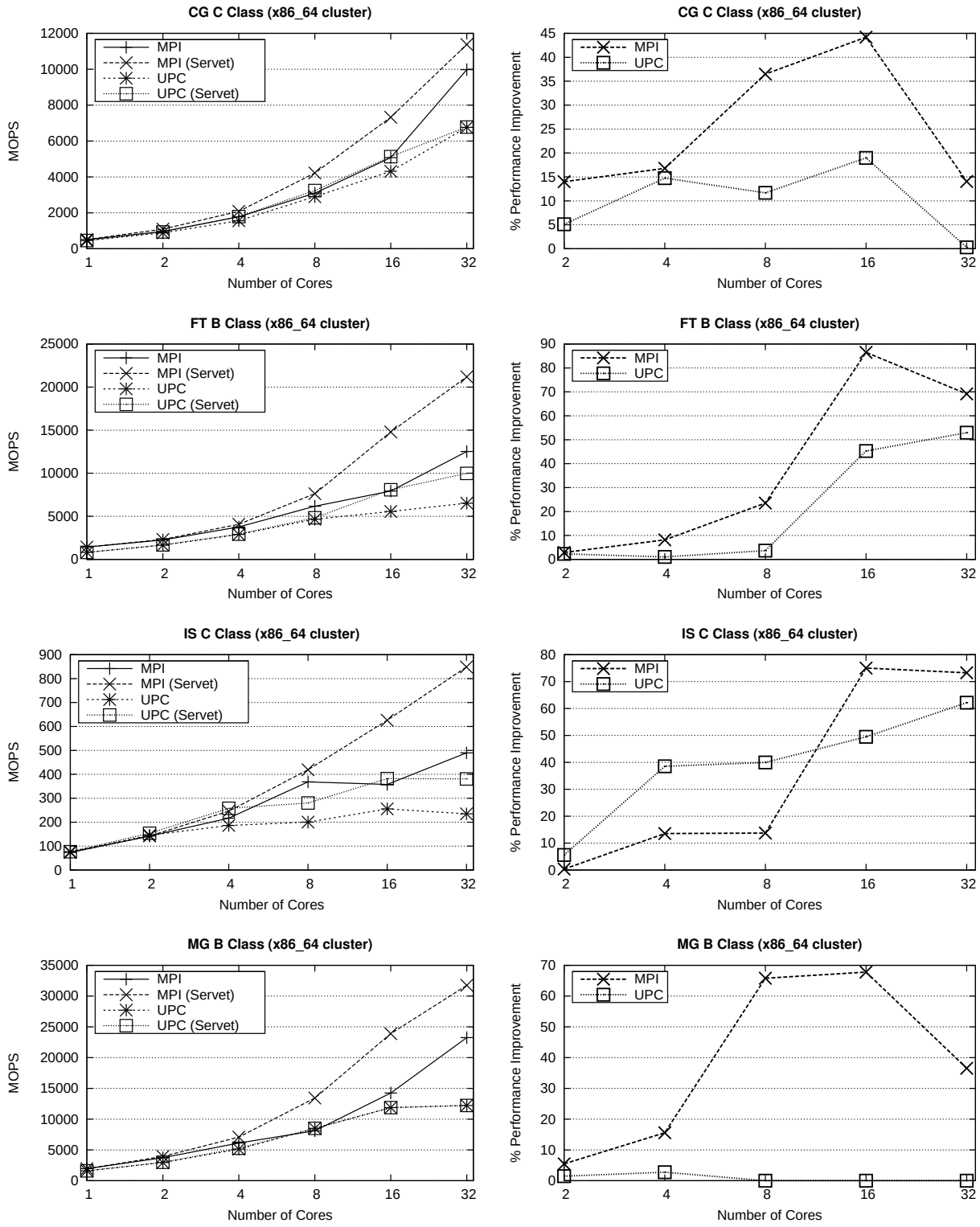

Figure 2: Impact of Servet on NPB performance (x86_64 cluster)

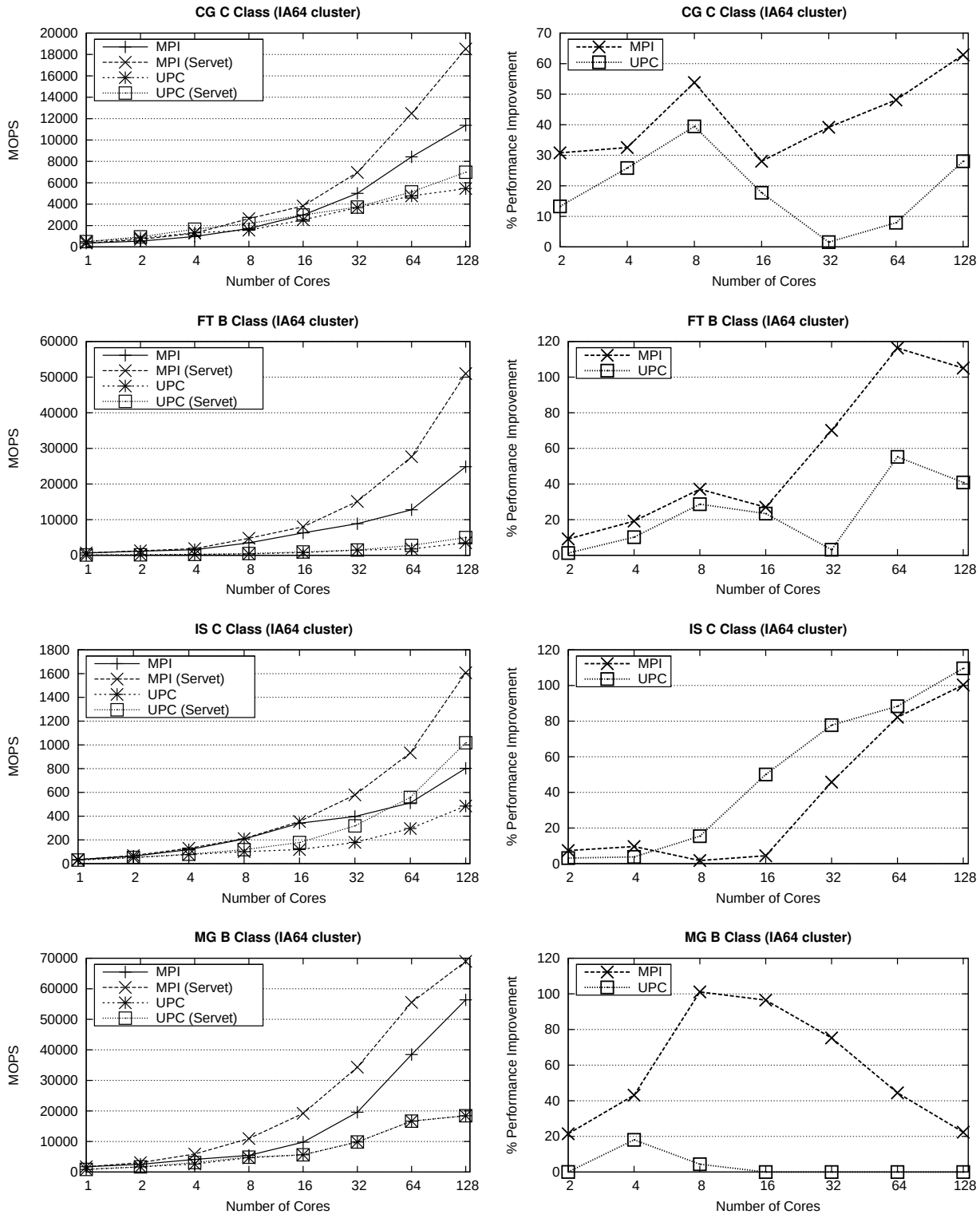

Figure 3: Impact of Servet on NPB performance (IA64 cluster)

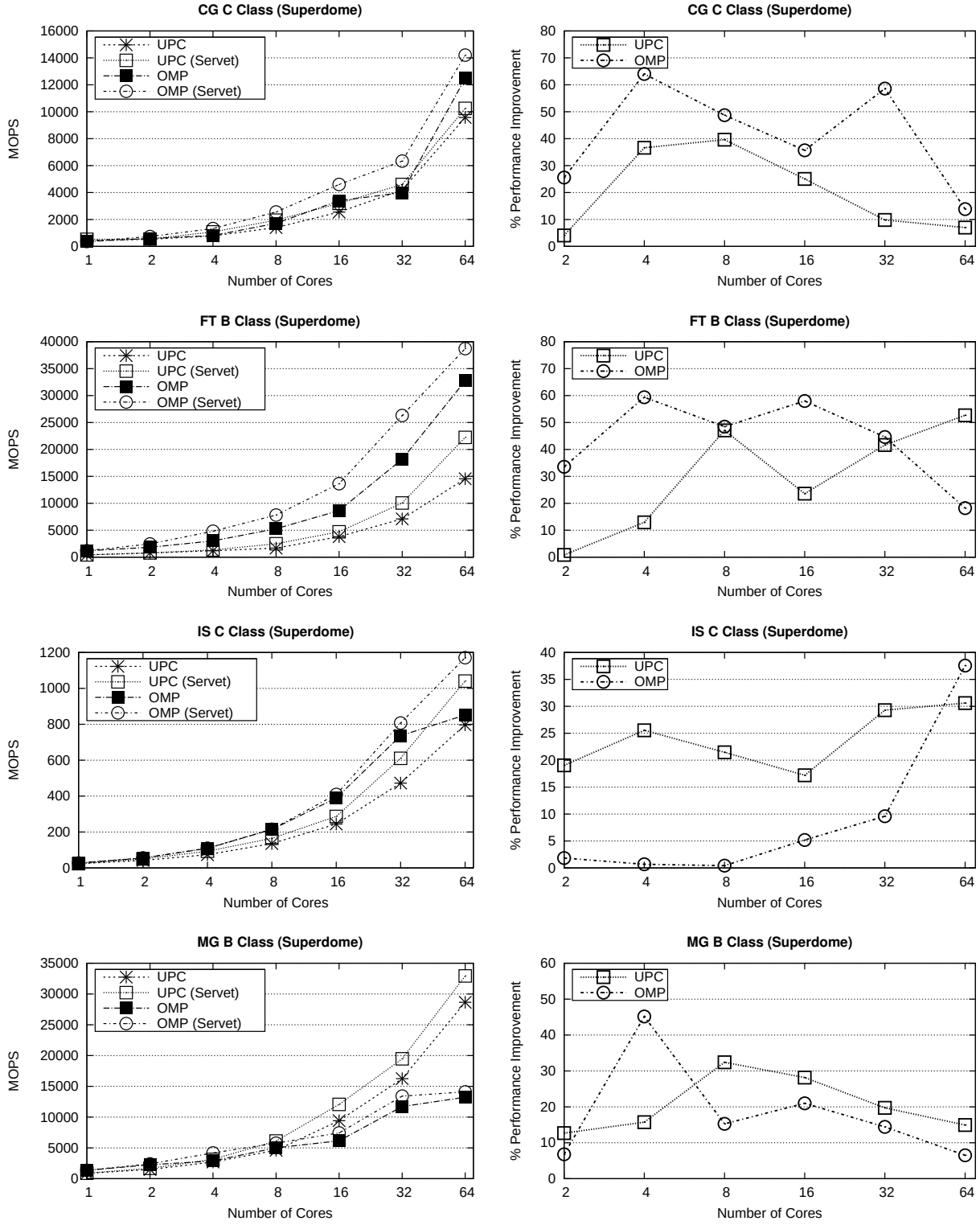

Figure 4: Impact of Servet on NPB performance (SMP Superdome)

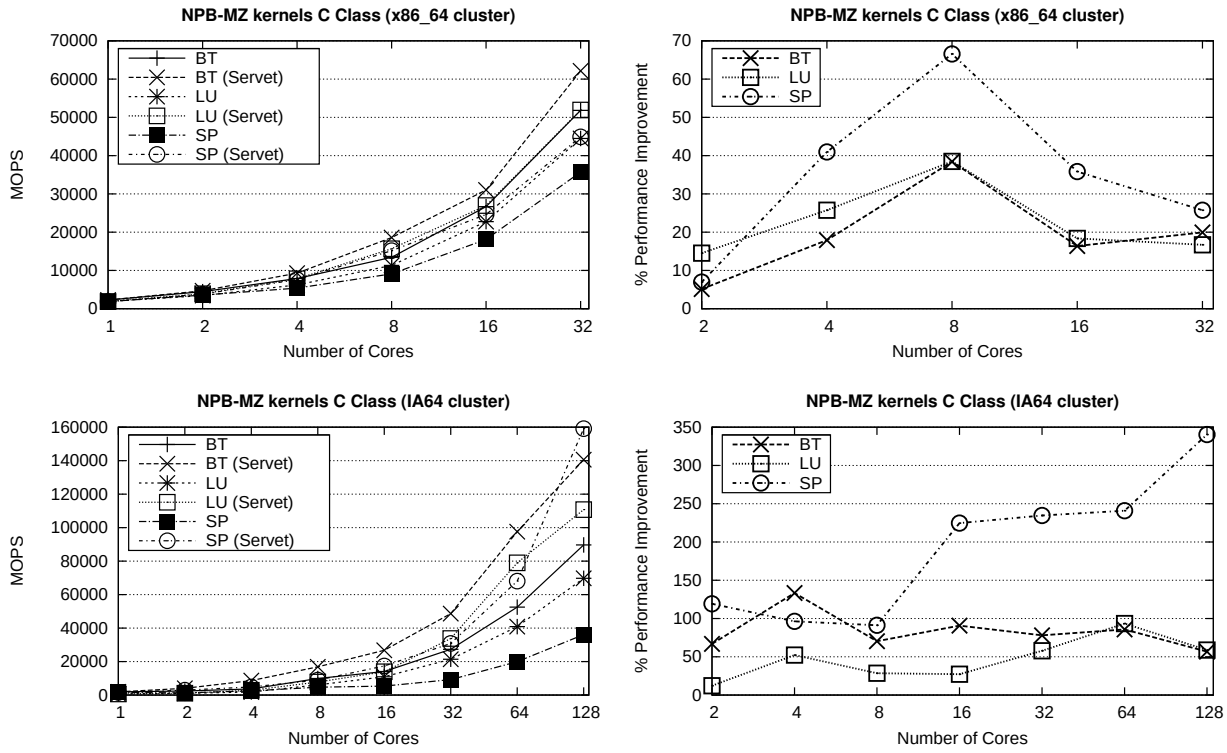

Figure 5: Impact of Servet on NPB-MZ performance (x86_64 and IA64 clusters)

Jorge González-Domínguez received the B.S. and the M.S. degrees in computer science from the University of A Coruña, Spain, in 2008 and 2010, respectively. He is currently a Ph.D. student in the Department of Electronics and Systems of the University of A Coruña. His main research interests are in the areas of PGAS programming languages and middleware for high performance computing. His homepage is http://www.des.udc.es/∼jgonzalezd.

Guillermo L. Taboada received the B.S., M.S., and PhD degrees in computer science from the University of A Coruña, Spain, in 2002, 2004 and 2009, respectively. He is currently an Assistant Professor in the Department of Electronics and Systems of the University of A Coruña. His main research interest is in the area of High Performance Computing (HPC), focused on high-speed networks, programming languages for HPC, cluster/grid/cloud computing and, in general, middleware for HPC. His homepage is http://www.des.udc.es/∼gltaboada.

Basilio B. Fraguela received the M.S. and the Ph.D. degrees in computer science from the University of A Coruña, Spain, in 1994 and 1999, respectively. He is an Associate Professor in the Department of Electronics and Systems of the University of A Coruña since 2001. His primary research interests are in the fields of performance evaluation and prediction, analytical modeling, design of high performance processors and memory hierarchies, and compiler transformations. His homepage is http://www.des.udc.es/∼basilio.

María J. Martín received the B.S. (1993), M.S. (1994) and Ph.D. (1999) degrees in physics from the University of Santiago de Compostela, Spain. Since 1997, she has been on the faculty of the Department of Electronics and Systems at the University of A Coruña, where she is currently an Associate Professor of Computer Engineering. Her major research interests include parallel algorithms and applications, cluster and grid computing and fault tolerance for message-passing applications. Her homepage is http://www.des.udc.es/∼mariam.

Juan Touriño received the B.S. (1993), M.S. (1993), and PhD (1998) degrees in computer science from the University of A Coruña, Spain. Since 1993, he has been on the faculty of the Department of Electronics and Systems, where he is currently a Full Professor of computer engineering and Director of the department. His main research interest is in the area of high performance computing and communications, covering topics such as languages and compilers, parallel and distributed algorithms and applications, and cluster/grid/cloud computing. His homepage is http://www.des.udc.es/∼juan.